University of Twente

Artificial Intelligence and Computer Security

Summary of the lectures

W. van den Brink
Based on the 2021 version of the Intelligent Interaction Design module

January 11, 2022

Contents

Week 1

Introduction to AI and Propositional Logic

Lecture 1

Introduction to AI

1.1 History of AI

The term artificial intelligence was first coined in 1956, at the Dartmouth Summer Research Project on Artificial Intelligence. This project "proceeded on the basis of the conjecture that every aspect of learning or any feature of intelligence can in principle be so precisely described that a machine can be made to simulate it".

Then started the golden years of AI (1956 - 1974), for example with the robot Shakey, an intelligent robot employing AI techniques. However, it also gave rise to inflated expectations of what AI might be capable of, and thus the disappointing realization of the problems we encounter when trying to develop AI. Common problems include trying to model common sense knowledge, infractability and the frame problem. This resulted in a so-called AI pendulum of AI winters and spring.

1.2 What is AI?

There are three common types of definitions for AI:

1. AI as a system (agent) interacting with the environment
2. AI as a collection of computational techniques for creating "intelligent" systems
3. AI as a multidisciplinary research field

This course focuses on the second definition. The HCI part of the module focuses on the third definition. We start, in weeks 1 to 3, with machine reasoning. In week 2, we briefly look at optimisation, where we apply algorithmic approaches. Finally, in weeks 4 to 6, we work with machine learning.

According to Russel and Norvig, an agent is anything that can be viewed as percieving its environment through sensors and acting upon this environment through actuators. A rational agent, then, is an agent that selects actions in order to maximize its performance measure, given evidence provided by the percepts and any built-in knowledge.

Lecture 2

Propositional Logic

2.1 Motivation

Propositional logic knows many applications. In AI, we use it to program reasoning and decision making. In order to design and implement the action selection of rational agents using propositional logic, we logically model the environment and the knowledge base *KB* of the agent. We then apply inference (reasoning) with this knowledge.

Example 1 (Wumpus World)
Wumpus World is a game where the agent must find a brick of gold in a world, represented by a grid. The grid contains a Wumpus, the gold brick and some pits. The agent should avoid the pits and the Wumpus, and use as few actions as possible. This environment has a few interesting properties:

- It is discrete: there is a finite number of distinct states and actions.
- It is static: the environment does not change during the game, unless the agent performs an action.
- There is a single agent: the Wumpus does not move.
- The environment is partially observable. The agent only percieves what is in its own square, not the full environment.

The following actuators are available to the agent:

- *Forward*: move one square forward in the direction the agent is facing, or stay put if there is a wall.
- *TurnLeft*, *TurnRight*: turn the direction the agent is facing.
- *Grab*: Pick up the gold when on the square the gold is at.
- *Shoot*: Fire an arrow in the direction the agent is facing, killing the Wumpus if it is in the path of the arrow. There is only one arrow.
- *Climb*: Climb out of the Wumpus cave. This actuator is only available on square $[1, 1]$.

The following sensors are available to the agent. The sensors are transmitted as a list of five items.

- *Stench*, in squares adjacent to the Wumpus.
- *Breeze*, in squares adjacent to a pit.
- *Glitter*, on the square with gold.
- *Bump*, when hitting a wall.
- *Scream*, when the Wumpus is killed.

Knowledge about logical reasoning steps needed for action selection in a rational agent is expressed using a knowledge representation language. New knowledge is derived by means of inferences: applying logical reasoning steps to derive new information from the existing information. The next action is derived from information and inferences about the state of the environment.

2.2 Syntax

In Propositional Logic, knowledge is represented by propositions. We discern the following types of propositions:

- **Atoms** or atomic sentences, e.g. $p, q, \text{gold}, \text{breeze}, \ldots$
- **Complex sentences** constructed from atomic sentences, using parentheses and logical connectives, e.g. $p, q, p \vee q, p \wedge q, \neg p, p \Rightarrow q, \ldots$
- **Literals**, either positive or negative, e.g. $p, q, \neg p, \ldots$

Definition 1
The following rules apply to sentences:

- If S is a sentence, then (S) is a sentence.
- If S is a sentence, then $[S]$ is a sentence.
- If S is a sentence, then $\neq S$ is a sentence (negation).
- If S_1 and S_2 are sentences, then $S_1 \wedge S_2$ is a sentence (conjunction).
- If S_1 and S_2 are sentences, then $S_1 \vee S_2$ is a sentence (disjunction).
- If S_1 and S_2 are sentences, then $S_1 \Rightarrow S_2$ is a sentence (implication).
- If S_1 and S_2 are sentences, then $S_1 \Leftrightarrow S_2$ is a sentence (biconditional or equivalence)
- Atoms are sentences: $p, q, r, s, t, \ldots$

Propositional Logic has its limits. There is almost no structure: complex properties can be represented by a single proposition p. One is allowed to give meaningful names to propositions, but it is not required. Modelling relations, like Mother(x, y) is not possible.

2.3 Semantics

We describe the meaning of a sentence in terms of its truth value (`true` or `false`, or equivalently 1 or 0). By assigning truth values to atoms, we can assign truth values to complex sentences. Such an assignment of truth values for the atoms of a sentence is called a model.

Definition 2
We say m is a model of a sentence α, or m satisfies α, if α is true in m. Notation: $m \models \alpha$.

Definition 3
M(α) is the set of all models of α: M$(\alpha) = \{m \mid m \models \alpha\}$

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p \Leftrightarrow q$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Table 2.1: Truth table for common operators

Definition 4 (Semantics of sentences)
Given a model m. Then:

- $\neg S$ is true iff S is false.
- $S_1 \wedge S_2$ is true iff S_1 is true and S_2 is true.
- $S_1 \vee S_2$ is true iff S_1 is true or S_2 is true.
- $S_1 \Rightarrow S_2$ is true iff S_1 is false or S_2 is true.
- $S_1 \Leftrightarrow S_2$ is true iff $S_1 \Rightarrow S_2$ is true and $S_2 \Rightarrow S_1$ is true.

This definition is displayed in Table 2.1.

Definition 5 (Entailment)
Given a knowledge base *KB* and a sentence α, we say that *KB* entails α if, in all models in which *KB* is true, also α is true. Notation: $KB \models \alpha$.

In symbols: $KB \models \alpha \Leftrightarrow \mathsf{M}(KB) \subseteq \mathsf{M}(\alpha)$.

Example 2 (Entailment of a knowledge base)
Let $KB = p \wedge q$, $\alpha = p$. Then $KB \models \alpha$.

Proof.

$$\left.\begin{aligned} \mathsf{M}(KB) &= \{(p=1, q=1\} \\ \mathsf{M}(\alpha) &= \{(p=1, q=1), (p=1, q=0)\} \end{aligned}\right\} \Rightarrow \mathsf{M}(KB) \subseteq \mathsf{M}(\alpha)$$

It follows that $KB \models \alpha$. □

Definition 6 (Logical equivalence)
$\alpha \equiv \beta$ if $\mathsf{M}(\alpha) = \mathsf{M}(\beta)$.

Definition 7 (Validity)
A sentence α is valid if it is true in all models. The sentence may also be called a tautology.

Definition 8 (Satisfiability)
A sentence α is satisfiable if it is true in some model, i.e. $\mathsf{M}(\alpha) \neq \emptyset$.

Theorem 1
$KB \models \alpha$ if $KB \wedge \neg a$ is unsatisfiable.

Proof. By contradiction. □

2.4 Reasoning

Definition 9
Theorem proving is the practice of applying rules to prove $KB \models \alpha$ without performing model checking. Notation: $KB \vdash \alpha$.

The rules for proving a theorem are notated by collecting facts, and adding a conclusion under a horizontal line. For example, for the well known rule Modus Ponens:

$$\frac{\alpha \Rightarrow \beta \quad \alpha}{\beta}$$

Definition 10
(Resolution rule) The idea of the resolution rule is to elliminate opposite literals of two disjuncts. The unit resolution rule is as follows:

$$\frac{p \vee q \quad \neg q}{p}$$

In general:

$$\frac{\begin{array}{l} l_1 \vee \cdots \vee l_k \\ m_1 \vee \cdots \vee m_n \end{array}}{l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \quad \vee \quad m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

where l_j and m_j are completementary literals:

1. $l_i = \neg m_j$ or
2. $\neg l_i = m_j$

on a syntactic level.

There is a technical aspect: we remove multiple copies of the same literal: $p \vee p \equiv p$.

Definition 11 (Conjunctive Normal Form)
A sentence is in CNF iff it is of the form:

$$S_1 \wedge S_2 \wedge \cdots \wedge S_n$$

In which each S_i is of the form:

$$(l_1 \vee l_2 \vee \cdots \vee l_l)$$

In which each l_i is a literal.

Theorem 2
Every syntactic correct sentence in Propositional Logic can be written in CNF.

Proof. Use the following procedure:

1. Replace $S_1 \Leftrightarrow S_2$ by $(S_1 \Rightarrow S_2) \wedge (S_2 \Rightarrow S_1)$.
2. Replace $S_1 \Rightarrow S_2$ by $\neg S_1 \vee S_2$.
3. Push $\neg$ inwards until it hits a literal. Use:

$$\begin{aligned} \neg(S_1 \wedge S_2) &= \neg S_1 \vee \neg S_2 \\ \neg(S_1 \vee S_2) &= \neg S_1 \wedge \neg S_2 \\ \neg\neg S_1 &= S_1 \end{aligned}$$

4. Distribute $\vee$ over $\wedge$ whenever possible:

$$(S_1 \wedge S_2) \vee S_3 = (S_1 \vee S_3) \wedge (S_2 \vee S_3)$$

The validity of this approach follows from the following equivalences:

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$	commutativity of $\wedge$
$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$	commutativity of $\vee$
$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$	associativity of $\wedge$
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$	associativity of $\vee$
$\neg(\neg\alpha) \equiv \alpha$	double-negation elimination
$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$	de Morgan
$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$	de Morgan
$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	distributivity of $\wedge$ over $\vee$
$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	distributivity of $\vee$ over $\wedge$

These equivalences can be proven using truth tables. □

Approach 1 (Proof by Resolution for Propositional Logic)
Given is a knowledge base KB and a model α. The goal is to prove $KB \vdash \alpha$.

1. Add $\neg\alpha$ to KB.
2. Write $KB \wedge \neg\alpha$ in CNF using Theorem 2.
3. Show that $KB \wedge \neg\alpha \vdash \bot$ by applying the resolution rule (Definition 10) until no new clause can be added anymore, or `false` is derived.

This approach is sound and complete:

- It is sound: if it is proven by resolution that α follows from KB, then $KB \models \alpha$.
- It is complete: if $KB \models \alpha$, then it can be proven by resolution that α follows from KB.

2.5 Programming with Logic

Not included. The author has completed module 8 `:-)`.

Week 2

Predicate Logic and Search

Lecture 1

Predicate Logic

1.1 Motivation & Syntax

Looking back on propositional logic (chapter 2):

- It is not very expressive.
- It does not fit to the way humans think and reason.
- Knowledge such as "Every adult is older than 17" is difficult to translate in Propositional Logic: it can only be done by enumeration.

We introduce (First-Order) Predicate Logic. Here, knowledge is represented by predicates, variables, quantifiers, and logical connectives. The syntax defines the following terms:

- **Constants**, like John, Mary, 2, UT, AI, ... They start with a capital letter.
- **Predicates**, like Parent, Daughter, Neighbor, $>$, Student, Takes, Smart, ..., which start with a capital letter and take one or more arguments between parenthesis.
- **Functions**, like Sqrt, Mother, Father, Teacher, ... They 'return' a result.
- **Variables**, like $x, y, z, \ldots$. They are placeholders for which we can choose values.
- **Connectives**: $\wedge, lor, \neg, \Rightarrow, \Leftrightarrow$.
- **Equality**: $=$.
- **Qualifiers**:
  - $\forall$: For all, the **universal quantifier**.
  - $\exists$: Exists, the **existential quantifier**.

Definition 12
An **atomic sequence** is either the application of a predicate with terms as arguments, or the equalization of two terms. In symbols:

- predicate $(\text{term}_1, \text{term}_2, \ldots, \text{term}_n)$
- $\text{term}_1 = \text{term}_2$

Definition 13
A **term** is either the application of a function with terms as arguments, a constant or a variable. In symbols:

- function $(\text{term}_1, \text{term}_2, \ldots, \text{term}_n)$
- Constant
- Variable

Sometimes, we use predicates as data structures, for example in Location (Wumpus, square $(1,3)$). Here, the predicate `Square` carries no special meaning, other than helping us see that the second argument of the predicate `Location` should be a square.

Definition 14
On top of the rule for sentences in propositional logic (Definition 1), the following rules apply to sentences in (first-order) predicate logic:

- If S is a sentence, then $\forall_x S(x)$ is a sentence.
- If S is a sentence, then $\exists_x S(x)$ is a sentence.

In these cases, x is like an argument for the sentence S.

1.2 Semantics

A model in (first-order) propositional logic contains objects, or domain elements, and the relations among them. The difference between the elements and relations in the domain and in the predicate logic are thin, but very important.

- Constant symbols are mapped to objects.
- Predicate symbols are mapped to relations.
- Function symbols are mapped to functional relations.

Usually, the names of the objects, relations and other symbols are the same in the domain as they are in the propositional equivalent. However, this need not be the case.

Definition 15 (Semantics of $\forall$)
$\forall_x P(x)$ is `true` in model m iff P is `true` in model m for every instantiation of x. In symbols:

$$\forall_x P(x) \equiv P(\mathsf{X}_1) \wedge P(\mathsf{X}_2) \wedge \cdots \wedge P(\mathsf{X}_n)$$

Where X_i is an instantiation of x. Remark that n may be infinite.

Definition 16 (Semantics of $\exists$)
$\exists_x P(x)$ is `true` in model m iff P is `true` in m for at least one instantiation of x. In symbols:

$$\exists_x P(x) \equiv P(\mathsf{X}_1) \vee P(\mathsf{X}_2) \vee \cdots \vee P(\mathsf{X}_n)$$

Where X_i is an instantiation of x. Remark that n may be infinite.

1.3 Unification

Before we can apply the resolution rule for predicate logic, we have to determine the equiality of two literals. Two given literals might not always be the same. By applying unification, we can make them equal.

Definition 17
A substitution is a set of one or more substitutions, which describe for the substituted variables and/or constants their old and new values. Notation: $\{\mathsf{Old}_1/\mathsf{New}_1, \mathsf{Old}_2/\mathsf{New}_2, \ldots, \mathsf{Old}_n/\mathsf{New}_n\}$

Definition 18
A unifier is a substitution which makes two literals equal.

Definition 19
We define the unification of two literals as follows:

1. If T_1 and T_2 are constants, then T_1 and T_2 unify if they are the same constant.
2. If T_1 is a variable and T_2 is any type of term, then T_1 and T_2 unify, and T_1 is instantiated to T_2 (and vice versa).
3. If T_1 and T_2 are complex terms then they unify if:
   (a) They have the same functor and arity, and

(b) All their corresponding arguments unify, and

(c) The variable instantiations are compatible.

Definition 20
The most general unifier (mgu) is a unifier for which it holds that every other unifier is an "extension". In other words: a mgu is a unifier with the least amount of variables replaced by concrete values.

1.4 Reasoning

A well known rule for theorem proving is Universal Instantiation:

$$\frac{\forall_x P(x)}{\text{Subst}\left(\{x/A\}, P(x)\right)}$$

Another well known rule for theorem proving is Existential Instantiation:

$$\frac{\exists_x P(x)}{\text{Subst}\left(\{x/A\}, P(x)\right)}$$

Intuition: we choose a value A which satisfies $P(x)$ to remove the existential quantifier.

We use mostly the same concepts for theorem proving in predicate logic as we do in propositional logic (Definition 6, Definition 7, Definition 8, Theorem 1). We add the following equivalence rules:

$$\begin{aligned}
\forall_x \neg P &\equiv \neg \exists_x P \\
\neg \forall_x P &\equiv \exists_x \neg P \\
\forall_x P &\equiv \neg \exists_x \neg P \\
\exists_x P &\equiv \neg \exists_x \neg P
\end{aligned}$$

Reasoning: to push a negation inwards, flip between universal or existential qualification.

Definition 21 (Resolution rule for predicate logic)
The idea of the resolution rule is to elliminate opposite literals of two disjuncts. It works as follows:

$$\frac{\begin{array}{l} l_1 \vee l_2 \vee \cdots \vee l_k \\ m_1 \vee m_2 \vee \cdots \vee m_n \end{array}}{\text{Subst}\left(\theta, l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \quad \vee \quad m_1 \vee \cdots \vee m_{j-1} \vee l_{j+1} \vee \cdots \vee m_n\right)}$$

Where $\text{Unify}\left(l_i, \neg m_j\right) = \theta$.

There is one technical aspect: we remove multiple copies of the same literal.

Approach 2
To transform a sentence in predicate logic to CNF, we use the following steps:

1. Replace all $A \Leftrightarrow B$ by $(A \Rightarrow B) \wedge (B \Rightarrow A)$
2. Replace all $A \Rightarrow B$ by $\neg A \vee B$
3. Move negation signs inwards, starting from the outside, until every negation sign is directly in front of a literal
4. Remove quantifiers
5. Distribute $\vee$ over $\wedge$ until you obtain CNF

Approach 3
To remove quantifiers from a sentence in predicate logic, we first eliminate existential quantifier by replacing the quantified variables by constant or functions using Skolemisation.

The general idea of Skolemisation: if $\exists_x P(x)$ is `true`, then $P(S)$ is also true. Here, S is a constant (the Skolem constant), which may not appear elsewhere in the proof. Intuition: a value for x is chosen. This only works when x is not within the scope of a universal quantifier.

When it is, the value of x depends on one or more other variables decided by the universal quantifier. We then replace x with a Skolem function $S(y)$ and eliminate the existential quantifier. In symbols:

$$\forall_{a,b,\dots}\exists_x P(x,y) \equiv \forall_{a,b,\dots} P(S(a,b,\dots),y)$$

When every existential quantifier is gone, every remaining variable is universally quantified. Thus, the universal quantifiers can be dropped, and all quantifiers are removed.

Approach 4 (Proof by resolution for predicate logic)
Given is a knowledge base *KB* and a model α. The goal is to prove $KB \vdash \alpha$.

1. Add $\neg\alpha$ to *KB*.
2. Write $KB \wedge \neg\alpha$ in CNF using Approach 2.
3. Show that $KB \wedge \neg\alpha \vdash \bot$ by applying the resolution rule (Definition 21) until no new clause can be added anymore, or `false` is derived.

This approach is sound and complete:

- It is sound: if it is proven by resolution that α follows from *KB*, then $KB \models \alpha$.
- It is refutation-complete: if $KB \models \alpha$, then it can be proven by resolution that α follows from *KB*.

  Resolution in predicate logic cannot be used to generate all logical consequences of a set of sentences.

1.5 Example Inference

Theorem 3
Given the following premises:

- Your grandparent is the parent of your parent
- John has a grandparent

We can conclude that John has a parent.

Proof. We use Approach 4.

We start by formalizing the premises:

1. $\forall_{X,y}\text{Grandparent}(x,y) \Rightarrow \exists_z \text{Parent}(x,z) \wedge \text{Parent}(z,y)$
2. $\exists_x \text{Grandparent}(x, John)$

Next, we formalize the conslusion, and negate it. We add this to our knowledge base.

3. $\neg\exists_x \text{Parent}(x, John)$

Remark that variables of the same name in different sentences are different variables. They can be instantiated to different values. As a precaution, we rename the variables, such that they are of different names in different sentences. Our sentences are now as follows:

1. $\forall_{X,y}\text{Grandparent}(x,y) \Rightarrow \exists_z \text{Parent}(x,z) \wedge \text{Parent}(z,y)$
2. $\exists_v \text{Grandparent}(v, John)$
3. $\neg\exists_w \text{Parent}(w, John)$

We then rewrite these sentences in CNF using Approach 2:

1. **Replace all $A \Leftrightarrow B$ by $(A \Rightarrow B) \wedge (B \Rightarrow A)$**

   We have no such sentences.

2. **Replace all $A \Rightarrow B$ by $\neg A \vee B$**

   This is required for the second rule:

$$\begin{aligned} &\forall_{X,y}\text{Grandparent}(x,y) \Rightarrow \exists_z \text{Parent}(x,z) \wedge \text{Parent}(z,y) \\ \equiv &\forall_{x,y} \neg \text{Grandparent}(x,y) \vee [\exists_z \text{Parent}(x,z) \wedge \text{Parent}(z,y)] \end{aligned}$$

3. **Move negation signs inwards, starting from the outside, until every negation sign is directly in front of a literal**

   We have no sentences where this is required.

4. **Remove quantifiers**

   First, we eliminate existential quantifiers:

   - $\exists_v \text{Grandparent}(v, John)$

     Simple Skolemisation works. We introduce a new constant G.

     $\text{Grandparent}(G, John)$.

   - $\neg\exists_w \text{Parent}(w, John)$

     Here, there is a negation before the quantifier. We can swap this quantifier for a universal quantifier by pushing the negation inwards.

     $\forall_w \neg \text{Parent}(w, John)$

   - $\forall_{x,y} \neg \text{Grandparent}(x,y) \vee [\exists_z \text{Parent}(x,z) \wedge \text{Parent}(z,y)]$

     We find that z is within the scope of both x and y. Thus, we need a Skolem function with two arguments:

     $\forall_{x,y} \neg \text{Grandparent}(x,y) \vee (\text{Parent}\,(x, S(x,y)) \wedge \text{Parent}\,(S(x,y), y))$

   Now, only universal quantifiers remain and we can drop them. Our sentences are as follows:

   1. $\text{Grandparent}(G, John)$
   2. $\neg \text{Parent}(w, John)$
   3. $\neg \text{Grandparent}(x,y) \vee (\text{Parent}\,(x, S(x,y)) \wedge \text{Parent}\,(S(x,y), y))$

5. **Distribute $\vee$ over $\wedge$ until you obtain CNF**

   Only the third sentence is not in CNF yet.

$$\begin{aligned} &\neg \text{Grandparent}(x,y) \vee (\text{Parent}\,(x, S\,(x,y)) \wedge \text{Parent}\,(S\,(x,y)\,, y)) \\ \equiv &[\neg \text{Grandparent}(x,y) \vee \text{Parent}\,(x, S\,(x,y))] \wedge [\neg \text{Grandparent}(x,y) \vee \text{Parent}\,(S\,(x,y)\,, y)] \end{aligned}$$

Finally, we apply resolution (Definition 21):

1. $\text{Grandparent}(G, John)$
2. $\neg \text{Parent}(w, John)$
3. $[\neg \text{Grandparent}(x,y) \vee \text{Parent}\,(x, S\,(x,y))]$
4. $[\neg \text{Grandparent}(x,y) \vee \text{Parent}\,(S\,(x,y)\,, y)]$
5. $\text{Parent}\,(S(G, John), John)$

   (1, 4, unify $\{x/G, y/John\}$, resolution)

6. $\neg$Parent $(S(G, John), John)$

   (2, 5, unify $\{w/S(G, John)\}$)

7. $\bot$

   (2, 6, resolution)

By contradiction, we have shown that $\exists_x$Parent$(x, John)$: John has a parent. $\square$

Lecture 2

Search

2.1 Defining a Search Problem

The path finding problem is defined as follows:

The **initial state** is the position of the agent at the start. The agent can then take **actions** to change the state of the agent. The available actions depend on the state of the agent. The **transition model** defines the result of an action.

With the **goal test**, one can test whether a given state is the goal state. The **path cost** assigns a numeric cost to each path.

The **solution** to the problem is a list of actions that leads from the initial state to the goal state. The **solution quality** then is measured by the path cost function.

Next, the **search space** is the set of all states reachable from the initial state by any sequence of action. The **search tree** or **search graph** is the structure formed by all the possible action sequences starting at the initial state. In this structure, the initial state is the root, the actions form the branches and the states are the nodes.

A **search strategy** expands nodes until either a solution is found, or no more states can be expanded.

There are four measures to evaluate search strategies:

- Time complexity
- Space complexity
- Completeness
- Optimality

Thus, it is favorable to have a search strategy which always finds the shortest path (if there is one), in the least amount of time and by consuming the least amount of memory.

2.2 Path Search Example - Representation

We consider an agent living in a very small grid-like world, with obstacles. The start and goal positions are given. We would like to find the shortest path from the start to the goal. We do this by reasoning about the path, not by exploring the world. The agent has a complete view of the relevant part of the environment. We then need to find an abstract representation of this environment in which our algorithms can work.

One such representation is the matrix E of size `width` $\times$ `height`, along with the following definitions:

- $E(i,j) = 0 \Rightarrow$ the cell at (i,j) is free.
- $E(i,j) = 1 \Rightarrow$ the agent is in cell (i,j).
- $E)i,j) = 2 \Rightarrow$ the cell at (i,j) is obstructed.

Every state is then $\mathsf{In}(i, j)$, with $1 \leq i \leq \texttt{width}$ and $1 \leq j \leq \texttt{height}$.

We define four possible actions `left`, `down`, `right` and `up`. The actions are applicable as follows:

- The action `left` is applicable in $\mathsf{In}(i, j)$ with $i > 1$, except when $E(i-1, j) = 2$.
- The action `down` is applicable in $\mathsf{In}(i, j)$ with $j > 1$, except when $E(i, j-1) = 2$.
- The action `right` is applicable in $\mathsf{In}(i, j)$ with $i > \texttt{width}$, except when $E(i+1, j) = 2$.
- The action `up` is applicable in $\mathsf{In}(i, j)$ with $j < \texttt{height}$, except when $E(i, j+1) = 2$.

The effect of the actions is as follows:

- Action `left`: $\mathsf{In}(i, j) \xrightarrow{\texttt{left}} \mathsf{In}(i-1, j)$
- Action `down`: $\mathsf{In}(i, j) \xrightarrow{\texttt{down}} \mathsf{In}(i, j-1)$
- Action `right`: $\mathsf{In}(i, j) \xrightarrow{\texttt{right}} \mathsf{In}(i+1, j)$
- Action `up`: $\mathsf{In}(i, j) \xrightarrow{\texttt{up}} \mathsf{In}(i, j+1)$

2.3 Basic Search Techniques

Algorithm 1 Tree search with breadth-first search.

```
frontier ← [(i, j), _] with E(i, j) = 1        ▷ _: extra information about the node.
loop
    if frontier = ∅ then
        return failure
    end if
    pop head node n from frontier
    if Goal state ∈ n then
        return solution
    end if
    expand n, add resulting nodes to the tail of frontier
end loop
```

Algorithm 2 Graph search with depth-first search.

```
frontier ← [(i, j), _] with E(i, j) = 1        ▷ _: extra information about the node.
closed ← []
loop
    if frontier = ∅ then
        return failure
    end if
    pop head node n from frontier
    add corresponding state to closed
    if Goal state ∈ n then
        return solution
    end if
    expand n
    if state ∉ closed ∪ frontier then add resulting nodes to the tail of frontier
    end if
end loop
```

Algorithm 1 is a simple strategy which first expands the root, then the root's successors, then their successors, and so on. It has two important issues:

- The search might consider multiple states repeatedly if the action space is cyclic, i.e. one can visit a state via multiple distinct action paths.
- The search is uninformed. No information about the goal state is used.

The first problem is solved in Algorithm 2 by keeping track of completely visited (closed) nodes.

Algorithm 2 is a simple strategy in which the deepest node is expanded before other nodes are considered.

The names graph search and tree search are confusing: both can be used on graphs and trees. The only difference is in the bookkeeping of completely visited nodes. Tree search is favorable in acyclic action paths, as the absence of the bookkeeping reduces both time and space complexity compared to graph search.

2.4 A* Search

In the previous section, we discussed several uninformed search strategies: graph and tree search with breadth-first and depth-first search. In this section, we present informed search strategies. These strategies use problem-specific knowledge beyond the problem itself, and are in general more efficient than uninformed search strategies. The lecture discusses the A* search strategy.

The general approach of A* is a *best-first* search. Either a tree or graph search algorithm is applied. However, the next node which is expanded is not based on its depth or breadth, but on an evaluation function $f(n)$. The evaluation function assigns a numeric value to the node, which represents the cost of the node. The node with the lowest cost is evaluated first.

Most such algorithms use a heuristic function $h(n)$ which estimates the cost of the cheapest path from the state at node n to the goal state, because it is not always possible to know this cost beforehand.

A* search then evaluates the node by combining the cost to reach the node n from the start state, denoted by $g(n)$, with the estimated cost to get from the node n to the goal, denoted by $h(n)$. Thus, the evaluation function is $f(n) = g(n) + h(n)$. Remark that the value of $g(n)$ is known. With this approach, the algorithm reduces the search space: nodes which are estimated to only make the cost worse are not considered.

Properties of A* search:

- A* is complete on finite graphs: it always finds a solution if there is one.
- However, A* will generally not explore the complete search space.
- A good heuristic function reduces the search space considerably.
- The tree version of A* is optimal if the heuristic function $h(n)$ is **admissible**. This is the case when $h(n)$ never overestimates the cost. In other words: $h(n)$ is optimistic.
- The graph version of A* is optimal if the heuristic function $h(n)$ is **consistent**. The function is consistent it the triangle inequality holds for the function. With this, we mean that the estimate for n is never greater than the estimate for any immediate neighbor (one-step successor) n', plus the cost of reaching n'. In symbols: $h(n) \leq \text{cost}(n, an') + h(n')$.
- A* is an example of blind commitment. The plan or route is not updated during the execution. Moving obstacles are not taken into account.

Week 3

Probabilistic Reasoning and Bayesian Networks

Lecture 1

Probabilistic Reasoning

1.1 Introduction and Motivation

Logic fits the way humans think and reason, but it only concerns reasoning about properties which are true or false. However, in general, we do not know if a statement is true or false, due to ignorance or intrinsic uncertainty. We can model uncertainty with probability theory.

1.2 Reasoning Using a Full Joint Probability Distribution

Definition 22
The joint probability is the probability of assigning a value to each random variable in a collection of random variables.

Notation: $P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$

Definition 23
A full joint probability distribution is a total collection of joint probability values.

Intuition: like a truth table, but with probabilities rather than truth values in the last column.

Definition 24 (Marginal probability)
A marginal probability is a probability over a subset of random variables. The notation is that of a joint probability, but with some variables missing between the parentheses. It is calculated by summing all probabilities for unknown variables:

$$P(X_1 = x_1, \ldots, X_k = x_k) = \sum_{x_{k+1} \in V_{k+1}} \cdots \sum_{x_n \in V_n} P(X_1 = x_1, \ldots, X_k = x_k, X_{k+1} = x_{k+1}, \ldots, X_n = x_n)$$

where $1 \leq k \leq n$ and V_i is the set of all possible values for the random variable X_i.

Definition 25 (Disjunction)

$$P(A \text{or} B) = P(A) + P(B) - P(A, B)$$

Definition 26 (Conditional probability)

$$P(A|B) = \frac{P(A, B)}{P(B)},\ P(B) > 0$$

Definition 27 (Probability distribution)
A probability distribution is a list of probabilities. It depicts the possible outcomes for all the values of a random variable. Notation: $\mathcal{P}(X_1, X_2, \ldots, X_n)$.

Corollary 1 (Sum of probabilities in a probability distribution)

$$\sum_{p \in \mathcal{P}} p = 1$$

Definition 28 (Conditional distribution)
A conditional distribution is like a probability distribution, but for conditional probabilities. Notation: $\mathcal{P}(A|B)$.

We can calculate the conditional distribution without knowing $P(A, B)$:

$$\begin{aligned} P(A|B) &= \left[[P(a,a), P(\neg a, b)] / \alpha_b, [P(a, \neg b), P(\neg a, \neg b)] / \alpha_{\neg b} \right] \\ \alpha_b &= P(a,a) + P(\neg a, b) \\ \alpha_{\neg b} &= P(a, \neg b) + P(\neg a, \neg b) \end{aligned}$$

This calculation assumes that A can only attain the values a and $\neg a$. The calculation can be generalized for random variables with more possible outcomes, or even continuous variables. This course only looks at discrete random variables.

1.3 Product, Chain and Bayes Rule

Definition 29 (Chain rule)
For n random variables $X_1, \ldots, X_n$:

$$\begin{aligned} P(X_1, \ldots, X_n) &= P(X_1, \ldots, X_{n-1}) P(X_n \mid X_1, \ldots, X_{n-1}) \\ &= P(X_1, \ldots, X_{n-2}) P(X_{n-1} \mid X_1, \ldots, X_{n-2}) P(X_n \mid X_1, \ldots, X_{n-1}) \\ &= \ldots \\ &= \Pi_{i=1} P(X_i \mid X_1, \ldots, X_{i-1}) \end{aligned}$$

Definition 30 (Bayes rule)

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

1.4 Independence and Conditional Independence

Definition 31
Two random variables A and B are independent if

1. $P(A \mid B) = P(A)$,
2. $P(B \mid A) = P(B)$, and
3. $P(A, B) = P(A)P(B)$

Lecture 2

Bayesian Networks

2.1 Overview

A **Bayesian network** is a simple, graphical notation for conditional independence assertions, and hence for a compact specification of full joint probability distributions.

The syntax is that of a mathematical graph $G(V, E)$ with the nodes being random variables, and the edges indicating direct influence between the random variables. The graph is directed and acyclic.

Definition 32
For a node X_i in a Bayesian network, its conditional probability is defined as follows:

$$P(X_i) = P(X_i \mid P_1, P_2, \ldots, P_n), \text{ with } \{P_1, P_2, \ldots, P_n\} \text{ the parents of } X_i$$

Consequently, if a node has no parents, its conditional probability is equal to the probability of the random variable.

Note
We will use Parents(X_i) to denote the parents of the node X_i. Formally,

$$\text{Parents}(X_i) = \{p \mid (p, X_i) \in E\}$$

From a Bayesian network, we can derive:

- The direct influence between variables,
- The indirect influence between variables,
- The conditional influence between variables (given other variables), and,
- The marginal independence between variables.

Furthermore, Bayesian networks allow us to reason *efficiently* about the probability distributions over latent (unobserved) variables.

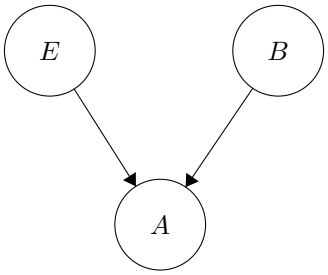
Definition 33
In a Bayesian network with variables $X_1, X_2, \ldots, X_n$, the joint distribution of all variables is defined as follows:

$$P(X_1, X_2, \ldots, X_n) = \Pi_{i=1}^{n} P(X_i \mid \text{Parents}(X_i))$$

The conditional distributions can be derived using marginalization.

2.2 Example 1: Burglary Alarm

Consider the following Bayesian network, modeling a burglary alarm:



The probability for $P(E)$ is defined as $P(e) = 0.001$, and, consequently, $P(\neg e) = 0.999$. In words: the chance of an earthquake happening is 0.1%. Likewise, the probability of a burglary happening is 2%. The network models the characteristic of a burglary alarm which triggers when a burglary happens, but also when an earthquake happens. The full joint probability is given below.

E	B	A	$P(A \mid E, B)$
e	b	a	0.97
e	b	$\neg a$	0.03
e	$\neg b$	a	0.29
e	$\neg b$	$\neg a$	0.71
$\neg e$	b	a	0.95
$\neg e$	b	$\neg a$	0.05
$\neg e$	$\neg b$	a	0.01
$\neg e$	$\neg b$	$\neg a$	0.99

Remark how the probabilities of A given the same values for E and B sum to 1. In the table, this is indicated with two without a horizontal border. Furthermore, observe that the alarm is not perfect: it might not trigger, when in fact it should.

We can now reason about this alarm. For example, what is the probability of a burglary if the alarm called you? We use Bayes theorem and the preliminaries given in the previous part to calculate this probability:

$$P(b \mid a) = \frac{P(b,a)}{P(a)} \qquad \text{(Bayes)}$$

$$\begin{aligned} P(b,a) &= P(e,b,a) + P(\neg e,b,a) && \text{(Marginalized probability)} \\ &= P(a \mid e,b)P(e)P(B) + P(a \mid \neg e,b)P(\neg e)P(b) \\ &= 0.97 \times 0.02 \times 0.001 + 0.95 \times 0.02 \times 0.999 \\ &= 0.019 \end{aligned}$$

$$P(a) = P(b,a) + P(\neg b,a) \qquad \text{(Marginalized probability)}$$

$$\begin{aligned} P(\neg b,a) &= P(e,\neg b,a) + P(\neg e,\neg b,a) && \text{(Marginalized probability)} \\ &= P(a \mid e,\neg b)P(e)P(\neg b) + P(a\neg e,\neg b)P(\neg e)P(b) \\ &= 0.29 \times 0.98 \times 0.001 + 0.01 \times 0.98 \times 0.999 \\ &= 0.0101 \end{aligned}$$

$$\begin{aligned} P(a) &= P(b,a) + P(\neg b,a) \\ &= 0.019 + 0.0101 = 0.0291 \end{aligned}$$

$$P(b \mid a) = \frac{0.019}{0.0291} \approx 0.6535$$

Next, we can calculate the probability of a burglary if the alarm went off *and* there was an earthquake:

$$P(b \mid a,e) = \frac{P(b,a,e)}{P(a,e)}$$

$$\begin{aligned} P(b,a,e) &= P(a \mid e,b)P(e)P(b) \\ &= 0.97 \times 0.02 \times 0.001 && = 1.94 \cdot 10^{-5} \end{aligned}$$

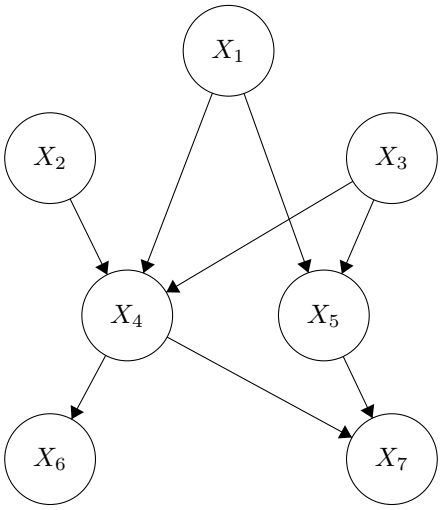
$$\begin{aligned} P(a,e) &= \cdot \\ &\approx 2.861 \cdot 10^{-4} \end{aligned}$$

$$\begin{aligned} P(b \mid a,e) &= \frac{1.94 \cdot 10^{-5}}{2.861 \cdot 10^{-4}} \\ &= 0.0678 \end{aligned}$$

Notice how $P(b) < P(b \mid a,e) \ll P(b \mid a)$.

2.3 Example 2: 7 Random Variables

Consider the following Bayesian network:



If the variables are binary, we would need to store $2^7 - 1 = 127$ parameters to know every joint probability: one for every value every value can attain, minus 1 we can derive when we know the rest.

However, with the network, we can reduce this to 21 parameters. Why? We only need to store 2^n parameters for a node with n parents. The number of required parameters then becomes $|\text{Parents}(X_1)| + |\text{Parents}(X_2)| + \cdots + \text{Parent}(X_7) = 1 + 1 + 8 + 4 + 2 + 4 = 21$. We can generalize this property:

Property 1
The number of required parameters for storing the joint probability of a Bayesian network with binary random variables $X_1, X_2, \ldots, X_n$ is calculated as follows:

$$\sum_{i=1}^{n} |\text{Parents}(X_i)|$$

2.4 Example 3: Entering college

Consider two characteristics of a person:

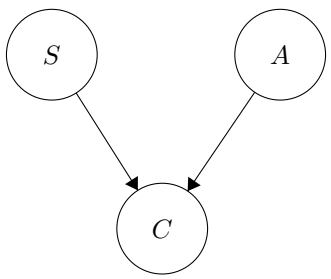
- Being smart, donated by the binary variable S, and,
- Being an athlete, denoted by the binary variable A.

Assume that 40% of the population is smart, and 10% of the population is an athlete.

Furthermore, let the binary variable C denote that a person entered college. Assume that being smart or being an athlete increase your chance of going to college. We formally define the probabilities as follows:

$P(C = c \mid A, S$	$A = a$	$A = \neg a$
$S = s$	0.91	0.90
$S = \neg s$	0.90	0.04

This example can be modeled in a Bayesian network as follows:



The full joint probability is defined as follows:

$$P(C, A, S) = p(C \mid A, S)P(A)P(S)$$

We can then perform several calculations on this model. For example, what is the probability that an athlete is smart?

$$P(s \mid a) = P(s) = 0.4$$

Next, what is the probability of meeting a smart person in college?

$$P(s \mid c) = \frac{P(c, s)}{P(c)} \approx 0.83$$

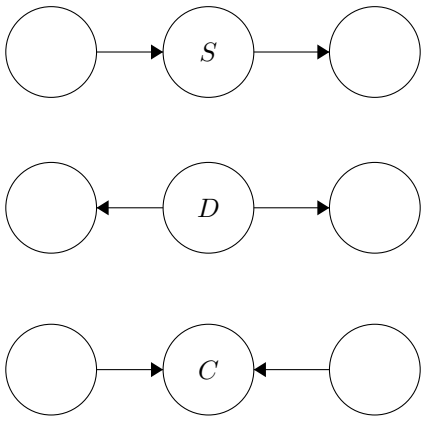
Finally, what is the probability of meeting a smart person in college, if that person is an athlete?

$$P(s \mid a, c) \approx 0.403$$

Pay attention to the way the informal questions are formalized.

2.5 Independence in Bayesian Networks

We distinguish the following types of nodes in a Bayesian network:



- A sequential node S is an intermediary between a cause and effect.
- A divergent node D is a common cause of two (or more) effects.
- A convergent node C is a common effect of two (or more) causes.

Definition 34 (Independence of nodes in a Bayesian network)
For the set of nodes A, B and C, $A \perp B \mid C$ if all paths from A to B are blocked.

To apply Definition 34, first determine all paths between A and B. A path is the list of edges between A and B, disregarding direction. Then, we say a path is blocked when:

- Two edges meet head-to-tail ($P \to W \to Q$) or tail-to-tail ($P \leftarrow W \to Q$) at a node which is in the observed set, or,
- Two edges meet head-to-head ($P \to W \leftarrow Q$) at a node which is not in the observed set, and none of whose descendants is in the observed set.

Here, the observed set is the set of all random variables which have attained a value.

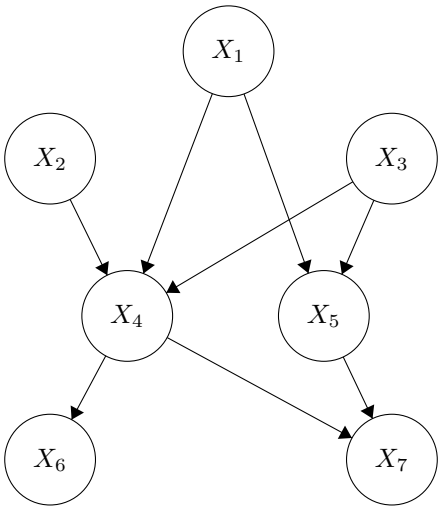
In symbols (more or less):

$$A \overset{+}{\rightarrow} W \overset{+}{\rightarrow} B \Rightarrow \begin{cases} A \text{ does not affect } B & \text{if } W \text{ is observed} \\ A \text{ affects } B & \text{otherwise} \end{cases}$$

$$A \overset{+}{\leftarrow} W \overset{+}{\rightarrow} B \Rightarrow \begin{cases} A \text{ does not affect } B & \text{if } W \text{ is observed} \\ A \text{ affects } B & \text{otherwise} \end{cases}$$

$$A \overset{+}{\rightarrow} W \overset{+}{\leftarrow} B \Rightarrow \begin{cases} A \text{ does not affect } B & \text{if } W \text{ is not observed} \\ A \text{ affects } B & \text{otherwise} \end{cases}$$

Consider again the example in section 2.3:



Because $X_1 \overset{+}{\rightarrow} X_4 \overset{+}{\leftarrow} X_2$, X_1 and X_2 are independent if X_4 is not observed. Otherwise, they become dependent. Likewise:

- X_1, X_2, X_3 are independent when X_4 and X_5 are unknown. They become dependent if either is known.
- X_6, X_7 are independent when X_4 is known.
- X_6 is independent of X_1, X_2, X_3, X_5, X_7 when X_4 is known.

While there is some intuition possible to guesstimate independence, it is recommended to (perhaps in your head) draw the paths between two nodes and really verify their independence before claiming it.

Week 4

Machine Learning

Lecture 1

Introduction to Machine Learning

1.1 Introduction to Machine Learning

Some aspects of intelligent systems in computer science:

- **Perception**: The investigation of the environment or of complex objects through a cognitive process.
- **Decision making**: The processing of incoming information, analysis of a situation, and the selection of possible actions, in order to achieve a goal.
- **Learning**: Data driven adaptation of the system. Achievable in multiple ways.

We classify machine learning problems on the spectrum of the amount of supervision associated with the learning process. The supervision indicates how much labels we give to the data beforehand.

- **Unsupervised** learning: we only supply data and give no labels.
- **Semi-supervised** learning: the data is only partly labeled.
- **Supervised** learning: the data is fully labeled.

Supervised learning gives the best results, but is more difficult to achieve, because it requires us to fully label the entire input data set, which is expensive and time-consuming.

With supervised learning, we make a distinction between classification and regression. Both processes attempt to assign a label to a new, unknown set of features. The difference is in the type of data. With **classification**, we attempt to predict a discrete label from the features. With **regression**, the predicted value is continuous.

Next, with semi-supervised learning, we partially label the data set. First, the model is trained on the labeled subset. Then, the model is extended by making predictions for the unlabeled data. Examples with high confidence are then accepted for the next training session. Thus, it is an iterative procedure.

Finally, in unsupervised learning, the data has no labels and we attempt to create labels by clustering the data. Data points are grouped by similarity, such that the points within each group are similar and the groups are dissimilar.

Unsupervised learning can be useful to determine the difference you are looking for, but it can be very difficult to understand.

1.2 Regularization

Supervised learning presents the following key difficulties:

- **Model selection**: deciding what model we use.
- **Data representation**: deciding how to represent the data for our model to consume.
- **Algorithm (parameters)**: deciding what algorithm to use, and how to use it.

Generally, we aim to train our models, such that the error rate *for new data* is low: we wish to increase the **generalization performance**. We do this with validation procedures.

We use the following notation for a data set:

$$D = \left\{ \{\xi^\mu, S^\mu\} \right\}_\mu = 1^P$$

Intuition. A data set is an ordered set of sets with cardinality P. By modifying μ and P we can take subsets of the data set (like one can slice an array). Every data point consists of a vector of features ξ^μ and the given label S^μ.

One strategy then, is to apply **n-fold cross validation**. We split the data (randomly) into n disjoint sets.

$$\begin{aligned} D &= \cup_{i=1}^n D^{(i)} && \text{all data} \\ D_{\text{train}}^{(i)} &= D \setminus D^{(i)} && \text{training data } (i) \\ D_{\text{test}}^{(i)} &= D^{(i)} && \text{test data } (i) \end{aligned}$$

The training is repeated n times, and the average training / test errors (and variances) are calculated. Then, by repeating the validation for different models, parameter settings and other factors which influence the results, the best system with respect to test errors can be selected.

Remark
Which n do we use in n-fold cross-validation? A larger n means a larger fraction of D is used in each training run. We have more estimates of E_{test} and we have smaller test sets. Furthermore, we have a higher computational effort. In the extreme case, we have $n = P$: every example but one is used for training, and the one example is used to test model, which we repeat P times. We call this **leave-one-out estimate**.

Remark
What about statistics? n results are *not* statistically independent. The training sets are highly overlapping, which makes it difficult to estimate variances with respect to the training set dependence.

When a model increases in complexity, it becomes better at classifying the training set. Consequently, it becomes better at classifying the test set. However, generally, the error rate for the test set is higher than the error rate for the training set. Furthermore, at a certain complexity, the model becomes too specific to the training set and does not correctly classify the test set anymore. This concept is called **over-fitting**: the error rate for the training set is low, while the error rate for the test set is high.

We wish to aim two competing goals:

- Low **bias**: the system only slightly deviates from the "true solution".
- Low **variance**: the system weakly depends on the training set.

The goals compete as follows:

- A low bias results in a complex model and over-fitting, which increases the variance.
- A low variance results in a simple model and under-fitting, which increases the bias.

1.3 Validation

The primary source for performance estimation for classification is the **confusion matrix**. The confusion matrix depicts how the model performs by measuring the amount of true or false positives and negatives.

		Actual	
		Positive	Negative
Predicted	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

We define the following measures, based on the values in the confusion matrix:

$$\begin{aligned}
\text{Accuracy} &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \\
\text{Precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\
\text{Recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \\
&= \text{Sensitivity} = \text{True Positive Rate} \\
\text{Specificity} &= \frac{\text{TN}}{\text{TN} + \text{FP}} \\
&= \text{True Negative Rate} \\
\text{F-score} &= 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}
\end{aligned}$$

Finally, notice that the confusion matrix can be extended for non-binary classifications simply by adding more rows and columns.

For regression, we wish to assess the difference between the predicted output and the target output.

We can calculate the **sum of squared errors**, which we want to minimize:

$$\text{E} = \frac{1}{2} \sum_{n=1}^{N} (y(X_n) - t_n)^2$$

Or we can calculate the probability of the predicted outputs given the target outputs, which we want to maximize.

For clustering, we define two internal measures.

The **cohesion** decides how closely the samples within a cluster are related. We calculate this with the **within sum squared errors (WSS)**:

$$\text{WSS} = \sum_i \sum_{x \in C_i} (x - m_i)^2$$

The **separation** decides how well separated one cluster is from the others. We calculate this with the **between cluster sum of squares (BSS)**:

$$\text{BSS} = \sum_i |C_i| \, (m - m_i)^2$$

with $|C_i|$ the size of the cluster C_i.

Lecture 2

Decision Trees

2.1 Decision Trees

Decision trees are a model for supervised learning. A tree-like model is used to fit the data. The trees are easy to understand and interpret.

A decision tree is a binary tree with splitting attributes as nodes and classes as leaves.

Two non-identical trees can fit the same data.

The process is as follows:

1. The training set is supplied to a tree induction algorithm, which induces a decision tree.
2. The resulting model is applied to a test set, which deduces the class.
3. The tree can then be evaluated for performance using a confusion matrix.

2.2 Entropy

Definition 35
Entropy is a measure of chaos in the dataset. Less entropy is better. More entropy requires more information to tell the class of a data point.

$$\mathsf{E}(S) = \sum_{i=1}^{c} -P_i \log_2 (P_i)$$

Intuition: sum the inverses (negatives) of the probability of a feature times the $\log_2$ of that probability.

Corollary 2
$\mathsf{E}(A) \leq 1$ for a binary attribute F.

Proof. The maximum entropy is reached when the attribute assumes both values equally often.

Then $\mathsf{E}(A) = 2 \cdot -\frac{1}{2} \log_2 \left(\frac{1}{2}\right) = 1$. □

Remark
The entropy of a data set is the entropy of the class.

2.3 Information Gain

Definition 36
Information gain is based on the entropy of a dataset. It tells us how much information we gain by knowing the value of an attribute.

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in A} \frac{|S_v|}{|S|} \text{Entropy}\,(S_v)$$

Intuition: consider every value assumed by the attribute. For every value, determine the entropy of the data set where the attribute takes on that value. Sum this entropy times the probability of that value.

2.4 Iterative Dichotomiser

The algorithm is as follows:

Input. A data set with attributes.
Output. A decision tree.

1. Find the attribute that provides the highest information gain within this data set.
2. Create a node using this attribute.
3. Sort out the data set to the corresponding split.
4. For each split data set: if all data points correspond to the same class, create a leaf node.
5. Otherwise, apply this algorithm to the split data set and insert the resulting tree here.

There are two base cases to prevent infinite recursion:

- If all the records in the current data set have the same class, then stop recursing.
- If all records have the same set of input attributes, then stop recursing.

2.5 Evaluating Classification Models

We estimate the performance of the model by splitting the data in a train and test set. The model is built on the train set and evalutated with the test. The model will trivially perform well on the train set, thus testing the performance with the train set is too optimistic.

Using accuracy as a performance metric fails sometimes. If the input data is "biased" in the sense that one class appears much more often than the other, a model might achieve a high success by blindly assigning the most probable class.

Using the confusion matrix works better. We then calculate the precision and recall for all the classes. We wish to maximize both.

2.6 Pruning Decision Trees

Decision trees can grow too deep and overfit. We fix this by removing splits that are not statistically significant. This has to happen after the tree has been built. This works as follows for binary classes, starting from the bottom:

1. Consider a node with p positive and n negative examples. Let p_1, n_1 and p_2, n_2 be the positive and negative examples for the two alternatives of the node.
2. We can state that this decision node does not provide an improvement if the ratio of positive and negative examples is (more or less) equal in the node and its children.

   In symbols:

$$\frac{p}{p+n} = \frac{p_1}{p_1+n_1} = \frac{p_2}{p_2+n_2}$$
$$\frac{n}{p+n} = \frac{n_1}{p_1+n_1} = \frac{n_2}{p_2+n_2}$$

3. Consider that $\frac{p}{p+n} = \frac{p_1}{p_1+n_1} \Rightarrow p_1 = p \cdot \frac{p_1+n_1}{p+n}$ by rewriting the terms.

   Define $\hat{p_1} = p \cdot \frac{p_1+n_1}{p+n}$. In a similar way, define $\hat{n_1}$, $\hat{p_2}$, and $\hat{n_2}$.

4. Calculate $\Delta = \frac{(p_1-\hat{p_1})^2}{\hat{p_1}} + \frac{(n_1-\hat{n_1})^2}{\hat{n_1}} + \frac{(p_2-\hat{p_2})^2}{\hat{p_2}} + \frac{(n_2-\hat{n_2})^2}{\hat{n_2}}$

   $\Delta \sim \chi^2_{d-1}$, with d the number of splits. Small values for Δ imply no rejection of the null hypothesis, and hence justify the pruning of the tree.

Week 5

Neural Networks and Reinforcement Learning

Lecture 1

Neural Networks

https://xkcd.com/1838/

Lecture 2

Reinforcement Learning

2.1 Motivation

Sometimes, the pair of configurations and labels becomes so large, that we cannot store them anymore. This requires a different learning paradigm. This lecture discusses reinforcement learning as a candidate for such problems.

The key concept in reinforcement learning is letting the agent learn from rewards or reinforcements.

2.2 Markov Decision Process

Definition 37
A Markov Decision Process consists of:

- a set of possible states, S,
- a set of actions A,
- for each state $s \in S$, a set of possible actions $a \in A(s)$,
- a probabilistic transition function $p(s' \mid s, a)$, and,
- a reward function $\mathrm{R}(s) : S \to \mathbb{R}$

A Markov Decision Process is a sequential decision problem in a fully observable stochastic environment, and it requires a Markovian transition model and additive reward. What does this mean?

- The agent takes one discrete step at a time.
- The environment is fully observable.
- Taking an action does not always result in the same outcome.
- Future states depend only on the current state, not on how the agent arrived at this state.
- The reward is the sum of all the rewards the agent received so far.

Furthermore, we have terminal states - states where the agent will stop when it reaches one. Every state has a reward, which can be positive or negative.

In a given MDP, we wish to find the optimal policy: a mapping $\pi : S \to A$ from states to actions.

An optimal policy π^* yields the highest expected utility, and balances risk and reward. The optimal policy is affected by the reward R of non-terminal states:

- Very low R: the agent wishes to terminate as quickly as possible.
- Low R: the agent wishes to terminate quickly.
- Small, but negative R: the agent avoids any risk.
- Positive R: the agent avoids any termination.

Rewards in an MDP are additive:

$$U_h\left[s_0, s_1, s_2, \ldots\right] = \mathrm{R}\left(s_0\right), \mathrm{R}\left(s_1\right), \mathrm{R}\left(s_2\right), \ldots$$

This equation is infinite. To assign a value to the equation, we introduce a discount factor $0 < \gamma < 1$, which decreases the value of a later reward, resulting in discounted rewards:

$$U_h\left[s_0, s_1, s_2, \ldots\right] = \mathrm{R}\left(s_0\right), \gamma\mathrm{R}\left(s_1\right), \gamma^2\mathrm{R}\left(s_2\right), \ldots$$

This equation is a geometric series and thus converges into a final value. We can use this property to present a formula for the utility of a state s, given a policy pi:

$$\mathrm{U}^{\pi}(s) = \mathbb{E}\sum_{t=0}^{\infty}\gamma^t\mathrm{R}\left(S_t\right)$$

We can now rank policies by expected rewards:

$$\pi_s^* = \underset{\pi}{\operatorname{argmax}}\,\mathrm{U}^{\pi}(s)$$

Remark that the optimal policy depends only on the current state. Thus, the only utility we are interested in is:

$$\mathrm{U}(s) = \mathrm{U}^{\pi^*}(s)$$

And thus, we can determine the best policy as the policy which selects the action with the best expected utility:

$$\pi^*(s) = \underset{a\in A(s)}{\operatorname{argmax}}\sum_{s'} P\left(s' \mid s, a\right)\mathrm{U}(s')$$

The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming optimal actions:

$$\mathrm{U}(s) = \mathrm{R}(s) + \gamma\max_{a\in A(s)}\sum_{s'} P\left(s' \mid s, a\right)\mathrm{U}\left(s'\right)$$

For n states, this results in n equations with n unknowns. The equations are non-linear and thus hard to solve. Iterative algorithms provide a solution to this problem.

2.3 Value Iteration

The value-iteration algorithm initializes utilities to zero. The utilities of terminal states are set to their respective rewards. Then, in an iterative fashion, the utilities are updated:

$$\mathrm{U}_{i+1}(s) \leftarrow \mathrm{R}(s) + \gamma\max_{a\in A(s)}\sum_{s'} P\left(s' \mid s, a\right)\mathrm{U}_i\left(s'\right)$$

This is repeated until the largest change in utility is smaller than a given threshold ϵ.

The algorithm converges to a unique solution. The speed of convergence depends heavily on γ.

2.4 Policy Iteration

The policy iteration algorithm starts with a random initial policy $\pi = \pi'$. Then, until there is no change anymore, for each $s \in S$, if there is an action such that

$$\left[\mathrm{R}(s) + \gamma \sum_{s' \in S} P\left(s' \mid s, a\right) \mathrm{U}\left(\pi, s'\right) \right] > \mathrm{U}(\pi, s)$$

then we set $\pi'(s)$ to a. Otherwise, the policy for this state is not changed.

2.5 Direct Utility Estimation

The aforementioned value iteration and policy iteration algorithms have a flaw: we oftentimes do not know the transition model nor the rewards a priori. We learn them from trials.

Given a fixed policy π, we learn the utility function without knowing $P\left(s' \mid s, a\right)$ or $R(s)$, by performing a few trials. Then we can define the utility as the (discounted) sum of rewards:

$$\mathrm{U}^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R\left(S_t, \pi(S_t), S_{T+1}\right) \right]$$

Intuition: follow the policy until you reach a terminating state, sum the rewards you found along the way, repeat then average.

2.6 Adaptive Dynamic Programming

The drawback of direct utility estimation is that it overlooks the recursive relation between the states. In Adaptive Dynamic Programming, we wish to learn the transition model $P\left(s' \mid a, a\right)$. Then, we can solve:

$$\mathrm{U}^{\pi}(s) = \mathrm{R}(s) + \gamma \sum_{s'} P\left(s' \mid s, \pi(s)\right) \mathrm{U}^{\pi}\left(s'\right)$$

For a fixed policy, this is a system of $|S|$ equations with $|S|$ unknowns, solvable with linear algebra or a linear algebra library.

Estimating the transition model is done with trials. Intuition: follow the policy until you reach a terminating state and repeat this a few times. Afterwards, approximate the probability of visiting a next state given a previous state and action by counting:

$$P\left(s' \mid s, a\right) = \frac{\text{times you visited } s' \text{ from } s \text{ with action a}}{\text{times you performed action } a \text{ at state } s}$$

2.7 Temporal Difference

ADP mandates frequent updates to the system of equations $\mathrm{U}(s)$ for every state s after each transition or observation. This is not optimal for a large number of states. It is furthermore useless for states that are unreachable from a specific state on which the transition was made.

2.8 Passive and Active Learning

With passive learning, the agent executes a fixed policy, limiting its freedom. In active learning, the agent needs to also find the optimal policy. This can be done by adapting ADP as follows:

- Estimate $P(s' \mid s, a)$ for more than one action.
- With the retrieved transition model, we can solve the MDP using value iteration.

- The action the agent should then choose is

$$\pi(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s' \in S} P(s' \mid s, a) U(s')$$

However, this approach updates the policy using partial information about the environment. This is a greedy approach. We need an approach that allows the agent to explore more of the environment. We wish to balance exploitation (converges faster) and exploration (slows learning). The solution is to introduce an exploration function $f(u, n)$:

$$U^+(s) \leftarrow \max_a f\left(\sum_{s'} P(s' \mid s, a)\Big[R(s, a, s') + \gamma U^+(s')\Big],\ N(s, a)\right)$$

$$f(u, n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

where $N(s, a)$ is the number of times the action a has been executed in state s. This enforces that a can be executed in s for N_e times.

Week 6

Machine Learning for Security

Lecture 1

Machine Learning for Security

1.1 Security

We describe the goals in security with the CIA paradigm: we wish to maintain

- **C**onfidentiality,
- **I**ntegrity, and,
- **A**vailability

of information and resources.

A **threat** is a potential violation of a security goal. An **attack** is an intentional violation of a security goal. Security is defined as the protection from intentional threats as opposed to the cost of the protection. A hacker is someone with an advanced understanding of computers and computer networks, and a willing ness to learn "everything". A black hat is a malicious hacker. An attacker need not be a hacker.

Security is maintained through a circular three step process of prevention, detection and analysis.

We discuss two challenges in security. First, there is an imbalance: an increasing amount of vulnerabilities, novel attack vectors and a high diversity of malware. Next, there are important bottlenecks: discovering new vulnerabilities, generating attack signatures and analyzing malware. This is currently a manual process done by human analysts.

We approach the second challenge with "smarter" security. By automating processes we can overcome shortcomings of manual actions and human processing.

1.2 Machine Learning

Machine learning is the automatic inference of dependencies from data. Dependencies are generalized, so that we can apply the learned dependencies to unseen data.

Formally, machine learning is an optmization problem, seeking a learning model θ, such that the expected error $\mathbb{E}(f_\theta)$ of the prediction function f_θ is minimized.

In practice, the expected error is unknown and we determine the empirical error $\mathbb{E}_n(f_\theta)$, based on n samples of training data.

A downside to this approach is the risk of overfitting. A remedy is k-fold cross validation.

1.3 Network Intrusion Detection System (NIDS)

An Intrusion Detection System monitors for attacks (an attempt to compromise confidentiality, integrity, or availability of a resource or information). There are several types of IDS; we focus on a system which monitors network traffic and uses machine learning to determine what traffic is benign.

First, we take a look at a traditional NIDS: the signature-based NIDS. It uses pattern detection (regular expressions, rules, contents of the payload) to determine whether traffic is an attack or not. This approach requires signatures from known attacks, and thus requires updating when new signatures are detected. This introduces scalability issues. Furthermore, the system is unable to detect novel or unknown attacks.

We quantify the performance of a machine learning based NIDS with the following metrics:

- **Effectivity**: the detection rate versus false alarms.
- **Efficiency**: the processing speed of the NIDS.
- **Robustness**: the resistance against evasion attacks, evolving attacks, and aging.

There are three approaches to a machine learning based NIDS. We can focus on malicious activity only, benign activity only, or on the differences between malicious and benign activity. We will be employing the second type, also called anomaly detection.

1.4 Anomaly Detection One-Class SVM

The One-Class Support-Vector-Machine is a standard algorithm for anomaly detection. It is effective, efficient and robust.

The general idea is to enclose the data in a hypersphere with minimum volume. Any data point outside the hypersphere is an anomaly.

We start with the input domain $D = \{d_1, d_2, \dots\}$.

A feature map ϕ: $D \to \mathbb{R}^N$ maps input items to a vector space $\mathbb{R}^N$ of real numbers:

$$d \mapsto \phi(d) = \langle\, \phi_1(d),\ \dots,\ \phi_N(d) \,\rangle \text{ for some } N > 0$$

Each $\phi_i(d)$ may exhibit a different scale. We apply Min-Max Normalization to scale each feature:

$$\overline{\phi_i}(d) = \frac{\phi_i(d) - \min_i}{\max_i - \min_i} \in [0, 1]$$

The dependencies are then not found by looking at the plain feature vectors, but rather the geometry and relationships induced by the vector space. We use kernels to map vectors to their geometry and relationships. Intuition: kernels map data to higher dimensions, where the data exhibits linear patterns, so we can linearly separate the data.

The One-Class SVM then becomes an optimization problem:

$$\mu_+ = \underset{\mu \in \mathbb{R}^N}{\operatorname{argmin}} \max_{1 \le i \le n} ||x_i - \mu||^2$$

with μ_+ the center of the hypersphere, and $X = \{x_1, \dots, x_n\}$ the feature vectors.

To increase robustness, we introduce softening: we allow for some local "slack" distance ξ_i of each feature vector x_i to the surface. This is done by introducing a global "softening" parameter $v \in [0, 1]$. The problem is now a constrained optimization problem:

$$\min_{R, \mu, \xi} R^2 + \frac{1}{vn} \sum_{i=1}^{n} \xi_i$$

subject to $||x_i - \mu||^2 \le R^2$ for all $i = 1, \dots, n$.

In machine learning terms, the learning model $\theta = (\mu_+)$, the prediction function $f_\theta(z) = ||z - \mu_+||^2$, and the prediction is performed using a threshold τ: predict z as "anomalous" if $f_\theta(z) \ge \tau$; otherwise predict z as "normal".